Story 1.4: HashiCorp Vault Integration

Status: drafted

Story

As a Developer,
I want HashiCorp Vault integrado para secrets management,
so that No hay credenciales hardcoded en código/config y cumplimos con security best practices.

Acceptance Criteria

AC1: Vault Docker Compose Service

Given El proyecto tiene Docker Compose configurado
When Agrego servicio de Vault al `docker-compose.yml`
Then

- Servicio `vault` configurado:
  - Imagen: `hashicorp/vault:1.15`
  - Puerto: `8200:8200`
  - Dev mode habilitado (`VAULT_DEV_ROOT_TOKEN_ID=dev-token-123`)
  - Capabilities: `IPC_LOCK` para mlock
  - Healthcheck configurado (`vault status`)
- Comando `docker-compose up -d vault` levanta Vault exitosamente
- Vault UI accesible en `http://localhost:8200/ui`

AC2: Spring Cloud Vault Dependencies

Given El proyecto tiene Spring Boot 3.2+
When Agrego dependencias de Vault
Then

- `pom.xml` incluye:
  - `spring-cloud-starter-vault-config` (Spring Cloud 2023.x)
  - `spring-vault-core` (managed por Spring Cloud)
- BOM `spring-cloud-dependencies` agregado con version 2023.0.0

AC3: Vault Bootstrap Configuration

Given Vault service running
When Configuro `bootstrap.yml` (carga antes de application.yml)
Then

- Vault URI: `http://localhost:8200`
- Authentication: `TOKEN` mode (dev)
- Token: `dev-token-123` (dev), `${VAULT_TOKEN}` (prod)
- KV engine: `secret` (KV v2)
- Application name: `signature-router`
- Fail fast: `true` (app no inicia si Vault no está disponible)

AC4: Secrets Initialization in Vault

Given Vault running en dev mode
When Inicializo secrets en Vault KV store
Then Secrets creados en path `secret/signature-router/`:

- `database.password` = "sigpass" (dev value)
- `kafka.sasl-jaas-config` = "" (placeholder para prod)
- `twilio.api-key` = "test-twilio-key-123"
- `twilio.api-secret` = "test-twilio-secret-456"
- `push-service.api-key` = "test-push-key-789"
- `biometric-sdk.license` = "test-biometric-license"
- Verificable con: `vault kv get secret/signature-router`

AC5: Replace Hardcoded Secrets in application-local.yml

Given Secrets almacenados en Vault
When Actualizo `application-local.yml`
Then

- Database password: Cambiado de `sigpass` a `${database.password}` (se carga desde Vault)
- Kafka SASL config: Placeholder `${kafka.sasl-jaas-config}` agregado
- Twilio API key: Placeholder `${twilio.api-key}` agregado
- No credenciales hardcoded en configs

AC6: VaultTemplate Bean Configuration

Given Spring Vault configurado
When Creo `VaultConfig.java`
Then

- Bean `VaultTemplate` configurado para programmatic access
- Método helper `getSecret(String path)` disponible
- Método helper `writeSecret(String path, Map<String, Object> data)` disponible
- VaultTemplate autowirable en services

AC7: Health Check for Vault

Given Vault configurado en Spring Boot
When Configuro Actuator health check
Then

- Endpoint `/actuator/health/vault` retorna `{"status":"UP"}`
- Health check verifica:
  - Conexión a Vault server exitosa
  - Token válido
  - KV engine accesible
- Si Vault está down, endpoint retorna `{"status":"DOWN"}` y app no inicia (fail fast)

AC8: Vault Secret Refresh (Dynamic Configuration)

Given Secrets almacenados en Vault
When Actualizo un secret en Vault
Then

- Spring Cloud Vault detecta el cambio (polling cada 60s en dev)
- Beans con `@RefreshScope` recargan valores
- No requiere restart de aplicación
- Logs indican refresh: `Refreshing beans with scope 'refresh'`

AC9: Environment-Specific Vault Configuration

Given Múltiples entornos (local, uat, prod)
When Configuro profiles en `bootstrap-{profile}.yml`
Then

- `bootstrap-local.yml`:

    - Vault URI: `http://localhost:8200`
    - Authentication: `TOKEN`, token: `dev-token-123`
- `bootstrap-uat.yml` (futuro):
    - Vault URI: `https://vault-uat.internal:8200`
    - Authentication: `KUBERNETES` (ServiceAccount token)
    - Role: `signature-router-uat`
- `bootstrap-prod.yml` (futuro):
    - Vault URI: `https://vault-prod.internal:8200`
    - Authentication: `KUBERNETES`
    - Role: `signature-router-prod`
    - TLS enabled

AC10: Vault Init Script (Docker Compose)

Given Vault running en dev mode
When Creo script `vault-init.sh` para seed secrets
Then

- Script ejecuta comandos Vault CLI:
    - `vault kv put secret/signature-router database.password=sigpass ...`
    - Crea todos los secrets de AC4
- Script ejecutable: `docker-compose exec vault sh /vault/scripts/vault-init.sh`
- Script idempotent (puede ejecutarse múltiples veces)

AC11: Integration Test with Vault

Given Spring Cloud Vault Test configurado
When Creo test de integración `VaultIntegrationTest.java`
Then

- Test usa Testcontainers Vault module
- Test verifica:
    - VaultTemplate puede leer secrets
    - Secrets inyectados vía `@Value("${database.password}")`
    - Health check Vault retorna UP
- Test pasa en `mvn verify`

AC12: Documentation & Security Guidelines

Given Vault infrastructure configurado
When Actualizo documentación
Then

- `README.md` actualizado con sección "Vault Setup":
  - Comandos Docker Compose para Vault
  - Comandos para inicializar secrets
  - Comandos para verificar secrets: `docker exec vault vault kv get secret/signature-router`
- `docs/development/vault-secrets.md` creado:
  - Vault architecture overview
  - Secret rotation strategy (futuro)
  - Troubleshooting (Vault sealed, token expired)
  - Production considerations (Kubernetes auth, TLS, seal/unseal)
- `CHANGELOG.md` actualizado con Story 1.4

Tasks / Subtasks

Task 1: Add Vault Service to Docker Compose (AC: #1)

- [ ] 1.1. Agregar servicio `vault` a `docker-compose.yml`:
  - Imagen: `hashicorp/vault:1.15`
  - Puerto: 8200
  - Dev mode: `VAULT_DEV_ROOT_TOKEN_ID=dev-token-123`
  - Capabilities: `IPC_LOCK`
- [ ] 1.2. Agregar healthcheck para Vault: `vault status`
- [ ] 1.3. Crear volumen para Vault scripts: `./vault/scripts:/vault/scripts`
- [ ] 1.4. Verificar: `docker-compose up -d vault`
- [ ] 1.5. Acceder Vault UI: `http://localhost:8200/ui` (token: `dev-token-123`)

Task 2: Add Spring Cloud Vault Dependencies (AC: #2)

- [ ] 2.1. Agregar BOM `spring-cloud-dependencies` version 2023.0.0 a `pom.xml`
- [ ] 2.2. Agregar `spring-cloud-starter-vault-config` dependency
- [ ] 2.3. Agregar property `<spring-cloud.version>2023.0.0</spring-cloud.version>`

◯ 2.4. Ejecutar: `./mvnw dependency:tree | grep vault`

◯ 2.5. Verificar: `spring-vault-core` incluido transitivamente

Task 3: Configure Vault Bootstrap (AC: #3)

◯ 3.1. Crear archivo `src/main/resources/bootstrap.yml`

◯ 3.2. Configurar `spring.cloud.vault.uri=http://localhost:8200`

◯ 3.3. Configurar `spring.cloud.vault.authentication=TOKEN`

◯ 3.4. Configurar `spring.cloud.vault.token=dev-token-123`

◯ 3.5. Configurar `spring.cloud.vault.kv.backend=secret`

◯ 3.6. Configurar `spring.cloud.vault.fail-fast=true`

◯ 3.7. Configurar `spring.application.name=signature-router` (para path discovery)

Task 4: Initialize Secrets in Vault (AC: #4)

◯ 4.1. Crear directorio `vault/scripts/`

◯ 4.2. Crear script `vault/scripts/vault-init.sh`:

```
vault kv put secret/signature-router \
  database.password=sigpass \
  kafka.sasl-jaas-config="" \
  twilio.api-key=test-twilio-key-123 \
  twilio.api-secret=test-twilio-secret-456 \
  push-service.api-key=test-push-key-789 \
  biometric-sdk.license=test-biometric-license
```

◯ 4.3. Dar permisos de ejecución: `chmod +x vault/scripts/vault-init.sh`

◯ 4.4. Ejecutar: `docker-compose exec vault sh /vault/scripts/vault-init.sh`

◯ 4.5. Verificar: `docker exec vault vault kv get secret/signature-router`

Task 5: Replace Hardcoded Secrets in application-local.yml (AC: #5)

◯ 5.1. Modificar `src/main/resources/application-local.yml`:

- Cambiar `spring.datasource.password: sigpass` → `${database.password}`

◯ 5.2. Agregar placeholders para Kafka SASL config (futuro):

- `spring.kafka.properties.sasl.jaas.config: ${kafka.sasl-jaas-config:}`

◯ 5.3. Verificar que no queden credenciales hardcoded con búsqueda: `grep -r "password.*:" src/main/resources/`

◯ 5.4. Documentar en comentarios: `# Loaded from Vault: secret/signature-router`

Task 6: Configure VaultTemplate Bean (AC: #6)

- ◯ 6.1. Crear `src/main/java/com/bank/signature/infrastructure/config/VaultConfig.java`
- ◯ 6.2. Autowire `VaultTemplate` bean (provisto por Spring Cloud Vault)
- ◯ 6.3. Crear método helper: `public String getSecret(String path)`
- ◯ 6.4. Crear método helper: `public void writeSecret(String path, Map<String, Object> data)`
- ◯ 6.5. Documentar JavaDoc con ejemplos de uso

Task 7: Configure Vault Health Check (AC: #7)

- ◯ 7.1. Verificar `spring-boot-starter-actuator` en `pom.xml` (ya incluido)
- ◯ 7.2. Configurar `management.health.vault.enabled=true` en `application.yml`
- ◯ 7.3. Exponer endpoint en `management.endpoints.web.exposure.include` (ya configurado)
- ◯ 7.4. Verificar: `curl http://localhost:8080/actuator/health/vault`
- ◯ 7.5. Test: detener Vault, verificar health check retorna DOWN y app falla en startup (fail-fast)

Task 8: Configure Vault Secret Refresh (AC: #8)

- ◯ 8.1. Configurar `spring.cloud.vault.config.lifecycle.enabled=true` en `bootstrap.yml`
- ◯ 8.2. Configurar `spring.cloud.vault.config.lifecycle.min-renewal=60s` (polling interval)
- ◯ 8.3. Crear bean de ejemplo con `@RefreshScope` para demo
- ◯ 8.4. Test manual: cambiar secret en Vault, esperar 60s, verificar bean refresca
- ◯ 8.5. Verificar logs: `Refreshing beans with scope 'refresh'`

Task 9: Configure Environment-Specific Vault Profiles (AC: #9)

- ◯ 9.1. Crear `src/main/resources/bootstrap-local.yml`:
  - Vault URI: `http://localhost:8200`
  - Authentication: TOKEN, token: `dev-token-123`
- ◯ 9.2. Crear `src/main/resources/bootstrap-uat.yml` (placeholder):
  - Vault URI: `https://vault-uat.internal:8200`
  - Authentication: KUBERNETES, role: `signature-router-uat`

- ◯ 9.3. Crear `src/main/resources/bootstrap-prod.yml` (placeholder):
  - Vault URI: `https://vault-prod.internal:8200`
  - Authentication: KUBERNETES, role: `signature-router-prod`, TLS enabled
- ◯ 9.4. Documentar diferencias en `docs/development/vault-secrets.md`

Task 10: Create Vault Init Script (AC: #10)

- ◯ 10.1. Crear `vault/scripts/vault-init.sh` (ya en Task 4.2)
- ◯ 10.2. Agregar shebang: `#!/bin/sh`
- ◯ 10.3. Agregar check idempotent: verificar si secrets ya existen antes de crear
- ◯ 10.4. Agregar logging: `echo "Initializing Vault secrets..."`
- ◯ 10.5. Documentar uso en README.md

Task 11: Create Integration Test with Vault (AC: #11)

- ◯ 11.1. Agregar dependency `org.testcontainers:vault` (test scope)
- ◯ 11.2. Crear `src/test/java/com/bank/signature/infrastructure/VaultIntegrationTest.java`
- ◯ 11.3. Configurar `@Testcontainers` con `VaultContainer` (dev mode)
- ◯ 11.4. Test method: `testVaultTemplateCanReadSecrets()`
  - Crear secret en Vault container
  - Leer con VaultTemplate
  - Verificar valor correcto
- ◯ 11.5. Test method: `testSecretsInjectedViaValueAnnotation()`
  - Autowire bean con `@Value("${database.password}")`
  - Verificar valor cargado desde Vault
- ◯ 11.6. Test method: `testVaultHealthCheckReturnsUp()`
- ◯ 11.7. Ejecutar: `./mvnw verify -Dtest=VaultIntegrationTest`

Task 12: Update Documentation (AC: #12)

- ◯ 12.1. Actualizar `README.md` con sección "Vault Setup":
  - Comandos Docker Compose para Vault
  - Comandos para inicializar secrets (`vault-init.sh`)
  - Comandos para verificar secrets: `docker exec vault vault kv get secret/signature-router`

- [ ] 12.2. Crear `docs/development/vault-secrets.md`:
    - Vault architecture overview (dev mode vs prod)
    - Secret rotation strategy (futuro - Vault dynamic secrets)
    - Troubleshooting (Vault sealed, token expired, connection refused)
    - Production considerations (Kubernetes auth, TLS, seal/unseal, HA)
    - Secret naming conventions
- [ ] 12.3. Actualizar `CHANGELOG.md` con Story 1.4 entry
- [ ] 12.4. Agregar comentarios JavaDoc en `VaultConfig.java` explicando uso

Dev Notes

Architecture Patterns

- **Secret Management:** HashiCorp Vault como single source of truth para secrets
- **Fail Fast:** App no inicia si Vault no está disponible (fail-fast=true)
- **Dynamic Configuration:** Spring Cloud Vault + @RefreshScope permite actualizar secrets sin restart
- **Environment Separation:** bootstrap-{profile}.yml para configuración específica por entorno

Source Tree Components

New Files:

- `src/main/resources/bootstrap.yml` (Vault bootstrap config)
- `src/main/resources/bootstrap-local.yml` (dev profile)
- `src/main/resources/bootstrap-uat.yml` (UAT profile placeholder)
- `src/main/resources/bootstrap-prod.yml` (Prod profile placeholder)
- `src/main/java/com/bank/signature/infrastructure/config/VaultConfig.java` (VaultTemplate bean)
- `src/test/java/com/bank/signature/infrastructure/VaultIntegrationTest.java` (integration test)
- `vault/scripts/vault-init.sh` (secret initialization script)
- `docs/development/vault-secrets.md` (Vault documentation)

Modified Files:

- `docker-compose.yml` (agregado servicio vault)

- `pom.xml` (agregadas dependencies Spring Cloud Vault)
- `src/main/resources/application-local.yml` (reemplazados hardcoded secrets con ${vault.placeholders})
- `src/main/resources/application.yml` (agregado management.health.vault.enabled)
- `README.md` (agregada sección Vault Setup)
- `CHANGELOG.md` (agregada entry Story 1.4)

Testing Standards

- **Integration Tests:** Testcontainers VaultContainer para tests aislados
- **Health Check Tests:** Verificar `/actuator/health/vault` retorna UP
- **Secret Injection Tests:** Verificar `@Value("${vault.secret}")` funciona correctamente
- **Fail Fast Tests:** Verificar app no inicia si Vault no disponible

Security Considerations

- **Dev Mode Only:** `VAULT_DEV_ROOT_TOKEN_ID` solo para desarrollo local (NUNCA en prod)
- **Kubernetes Auth:** Producción debe usar Kubernetes ServiceAccount authentication
- **TLS:** Vault en prod DEBE usar TLS (https://vault-prod.internal:8200)
- **No Hardcoded Secrets:** PROHIBIDO hardcodear secrets en código/configs
- **Secret Rotation:** Futuro: implementar dynamic secrets con TTL

Project Structure Notes

Alineado con estructura hexagonal:

- `VaultConfig.java` en `infrastructure/config/` (correcto - Vault es infraestructura)
- Domain layer NO debe depender de Vault (secrets inyectados vía ports)
- Application layer recibe secrets vía `@Value` o constructor injection

References

- **Epics Document**: Story definition, AC, technical notes
- **Tech Spec Epic 1**: Vault 1.15 especificado
- **Architecture - Security**: PseudonymizationService usa `secretManager.getSecret()`
- **PRD - Security Requirements**: NFR41 (encryption at rest), NFR42 (secret rotation)
- **Spring Cloud Vault Docs**: https://docs.spring.io/spring-cloud-vault/docs/current/reference/html/
- **Vault Docker Image**: https://hub.docker.com/_/vault

Dev Agent Record

Context Reference

Agent Model Used

Claude Sonnet 4.5

Debug Log References

Completion Notes List

File List

Created:

Modified:

Deleted:

Change Log

Date	Author	Change
2025-11-26	BMAD SM Agent	Story 1.4 draft created: HashiCorp Vault Integration for secrets management